УДК 004.054 DOI: 10.34824/VKNIIRAN.2022.11.3.006

ОБНОВЛЕНИЕ ФУНКЦИОНАЛА JAVASCRIPT ЧЕРЕЗ ECMASCRIPT 6

© Гишлаков Сайфулла Умарович (a), Джангаров Ахмед Идрисович (b)

(a) Чеченский государственный университет им. А.А. Кадырова, Российская Федерация, г. Грозный; студент 3 курса Института математики, физики и информационных технологий, sgishlakaev@mail.ru

(b) Чеченский государственный университет им. А.А. Кадырова, Российская Федерация, г. Грозный; ассистент кафедры программирования и инфокоммуникационных технологий, dzhangarov1995@gmail.com

Аннотация. ECMAScript 6 – это стандарт который определяет каким будет язык JavaScript, язык общего назначения ECMAScript имеет свои баги и недочеты, но обновления происходят регулярно, и язык улучается. Это обновление имеет глобальный характер для скриптовых языков, так как привнесло очень много полезных вещей, от синтаксического сахара до изменений в методах и логике.

***Ключевые слова*:** веб, JavaScript, ECMAScript, ES6, баги, ошибки, обновления, фронт-энд.

UPDATING JAVASCRIPT FUNCTIONALITY VIA ECMASCRIPT 6

© Gishlakaev Sayfulla Umarovich (a), Dzhangirov Ahmed Idrisovich (b)

(a) Chechen State University A.A. Kadyrov, Russian Federation, Grozny; 3rd year student of the Institute of Mathematics, Physics and Information Technologies, sgishlakaev@mail.ru

(b) A.A. Kadyrov Chechen State University, Grozny, Russian Federation; Assistant of the Department of Programming and Infocommunication Technologies, dzhangarov1995@gmail.com

Abstract. ECMAScript 6 is a standard that defines what the JavaScript language will be, the general–purpose ECMAScript language has its own bugs and shortcomings, but updates occur regularly, and the language is being improved. This update is global in nature for scripting languages, as it has brought a lot of useful things, from syntactic sugar to changes in methods and logic.

Key words: web, JavaScript, ECMAScript, ES6, bugs, errors, updates, frontend.

ECMAScript 6 на данный момент не последняя версия, она выпущена 2015 года, на данный момент последняя версия – это 13 версия, выпущенная в июне 2022 года. Версии с

7 по 13 почти одинаковы и мало чем отличаются, в данной статье мы рассмотрим изменения именно в ECMAScript 6. Версия 6 разрабатывалась около 10 лет, за этот большой срок разработки язык был сильно изменен, добавлено громадное количество новый функций. Так как срок перехода с 5 версии на 6 был огромен, ECMAScript 6 – считается самой важной обновой языка. Чтобы понять на сколько увеличили функционал языка стоит обратить внимание на объём документа спецификации языка, который увеличился примерно в 3 раза.

Начнем рассмотрение со стрелочных функций, это обновление сильно сократила код и повысила читаемость кода.

Например, есть стандартная функция с ES5.

```
const name = function myfunc() {
console.log("Hello from myfunc");
};
```

Она в стандарте ECMAScript 6 имеет вышенаписанный вид, это можно сократить до

```
const myFunc = () => {
console.log("Hello from myFunc");
}
```

можно убрать ключевое слово function как мы видим, но также можно пойти и дальше в сокращении.

Если тело функции состоит из одной строки, можно опустить фигурные скобки:

```
const myFunc = () => console.log("Hello from myFunc");
```

Если у *нас* нету параметра можно убрать круглые скобки:

```
const myFunc = console.log("Hello from myFunc")
```

Благородя этим сокращениям код становиться лаконичным.

Не смотря на эти незначительные изменения, функции работают одинаково.

ECMAScript 6 добавил возможность разграничения областей в работе с классами, а именно добавила ключевое слово this. В JavaScript ключевое слово this помогает обратиться к текущему полю в классе при передаче его через методы различные:

- при передаче через базовый метод класса, который объявляется при инициализации каждого экземпляра класса – через конструктор.
- при передаче через метод set.

Ключевое слово this может иметь различные ссылочные значения при определенных обстоятельствах:

- внутри метода при вызове this, обращение идет к объекту от которого вызван метод.
- в публичном поле, ключевое слово this не ссылается к конкретному отдельному объекту, а ссылается к публичному объекту.
- привязанность this к определенному объекту можно изменить через такие методы как call() и apply().

Одним из немаловажных добавлений можно считать способы выделения памяти под переменные, функции. В ES6 добавили обновленные способы создания переменной let и константу const. Между ними есть разница, например, если разница между var let и const очевидна, она заключается в том, что const нельзя переприсвоить, его значение указывается в момент инициализации константы.

```
const value = 3; // это значение больше нельзя менять
value = 4; // это вызовет ошибку
```

Отличие между let и var трудно заметить на первый взгляд. Различия заключаются в том, что var – это старый способ инициализировать переменную, в то время как let пришел с ES6. Let решил проблему видимости областей.

Например, создадим переменную внутри функции и попытаемся обратиться к ней за пределами этой функции, все это будет с помощью var;

```
function someFUnc () {
var value  = new value()
return value
                        }
someFUnc ()
console.log(value) // вызовет ошибку
```

Мы видим, что пока что все работает отлично так, как и должно. Так и должно быть разграничение на локальные и глобальные переменные. Но все начинает ломаться в более сложных примерах.

Более сложный пример будет заключаться в том, что мы возьмем два массива с ценами prs и со скидками skidk и вернем общий массив где будут учитываться скидки.

```
function skidkprs (prs, skidk) {
var skidked = []
for (var i = 0; i < prs.length; i++) {
var skidkedprs = prs [i] * (1 - skidk)
var finalprs = Math.round(skidkedprs * 100) / 100
skidked.push(finalprs)
  }
return skidked
}
```

Код в целом понятный и очевидный, но обратите внимание на переменные которые объявлены внутри цикла, а именно на: var skidkedprs; var finalprs и на переменную которая в качестве счётчика var i, в других языка программирования эти переменные не должны были бы быть доступны за пределами цикла, но если их объявлять с помощью var то они получаются доступными.

```
function skidkaPrs (prs, skidka) {
var skidkaed = []
for (var i = 0; i < prs.length; i++) {
var skidkaedPrs = prs[i] * (1 - skidka)
var finalPrs = Math.round(skidkaedPrs * 100) / 100
skidkaed.push(finalPrs)
  }
console.log(i)
console.log(skidkaedPrs)
console.log(finalPrs)
return skidkaed
}
```

Этот код на удивление скомпилируется и не вызовет никаких ошибок, несмотря на то, что в нем вызываются переменные, которые по сути являются локальными. Ниже мы видим, что через консоль вызвали локальные переменные, и они отобразятся на *консоли* если вызвать эту функцию с параметрами.

В других языках это вызвало бы ошибку, но тут все выведется на экран.

У нас нет нужды иметь доступ к переменным var skidkedprs; var finalprs; var i, так как они находятся внутри цикла и нигде больше не используются. Имея к ним доступ, мы можем усложнить код для других читателей нашего кода, особенно это проблема прослеживается в командной разработке, где пытаются код сделать максимально понятным и лаконичным. Наличие доступа может мешать и вам самим, когда в списке переменных будут выскакивать ненужные переменные из циклов.

Мы выяснили на примерах, что у переменной, инициализированной через устаревшей способ Var доступ функциональный, в зависимости от функций, то есть внутри функций либо же методов. А вот современный метод let лишен этих недостатков, так как его область видимости заключается в фигурных скобках, то есть между {}.

Например, код который работал раньше с var перепишем на let:

```
function skidkaPrs (prs, skidka) {

let skidkaed = []

for (let i = 0; i < prs. length; i++) {
let skidkaedPrs = prs[i] * (1 - skidka)
let finalPrs = Math.round(skidkaedPrs * 100) / 100
 skidkaed. push (finalPrs)
}

console.log(i) // будет ошибка
console.log(skidkaedPrs) // будет ошибка
console.log(finalPrs) // будет ошибка
```

```
return skidkaed
}
skidkaPrs ([1034, 3245, 532], .3) // будет ошибка
```

Тут уже интуитивно понятно, почему ошибка, потому что при вызове console.log будет ошибка, так как нельзя обращаться к локальным переменным.

В ES6 есть множество изменений, о которых не упомянуть в одной статье, это и классы, и промисы, и конструкторы, и ещё много чего.

Несмотря на то, что ECMAScript тру программисты, считают забагованным и не походящим для нормальной строгой разработки, он все-таки дополняется и изменяется с каждым годом, так как его поддержка продолжается. В настоящее время уже исправлены множество ошибок и проблем языка, его можно использовать самостоятельно для некоторых проектов.

ЛИТЕРАТУРА

1. Николас З. ECMAScript 6 для разработчиков. 2017 г.
2. Прасти Н. Введение в ECMAScript 6. 2016 г.
3. Марейн Х. Выразительный JavaScript. Современное веб-программирование. 3-е издание. 2020 г.
4. Рыбаков Е. JavaScript и Node.js для Web-разработчиков. 2022 г.
5. Черный Б. Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. 2021 г.

REFERENCES

1. Nicholas Z. ECMAScript 6 for developers. 2017
2. Prasty N. Introduction to ECMAScript 6. 2016
3. Maraine H. Expressive JavaScript. Modern web programming. 3rd edition. 2020
4. Rybakov E. JavaScript and Node.js for Web developers. 2022
5. Black B. Professional TypeScript. Development of scalable JavaScript applications. 2021